



DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
AIR UNIVERSITY

PROJECT REPORT

OPERATING SYSTEM

Instructors Name

Sir Ahmed Mujtaba

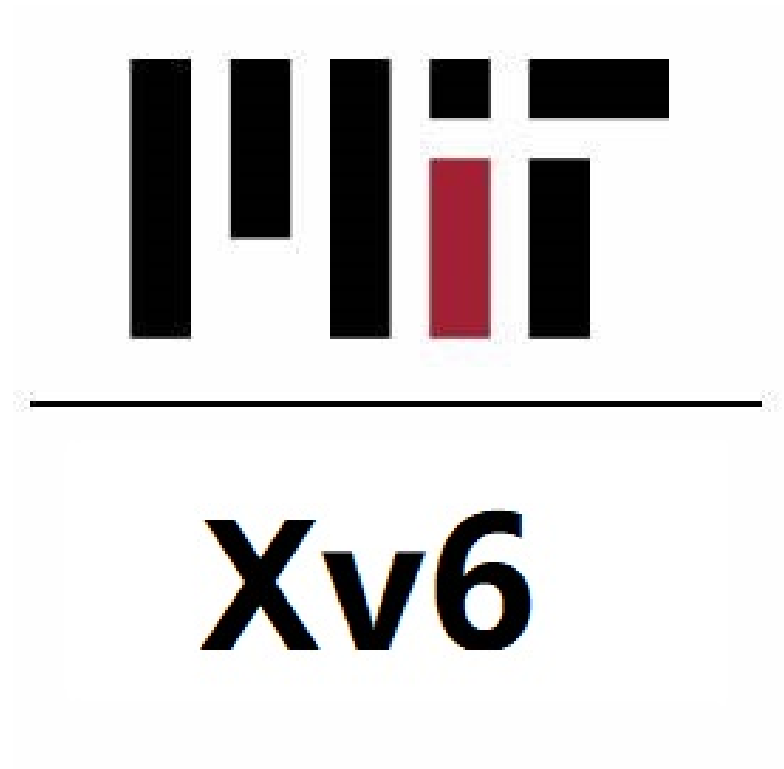
Sir Najam Dar

Submitted by

Muhammad Burhan Ahmed (210287)
Muhammad Bilal Ijaz (210316)
Agha Ammar Khan (210300)

23/12/2023

Complex Engineering Problem



Adding Preemptive Priority Scheduler to XV6

ABSTRACT

The goal of this project is to introduce a priority-based method to process scheduling in an operating system, therefore enhancing it. Two of the main goals are to develop a preemptive priority scheduler and modify the Process Control Block (PCB) structure such that each process has a priority field. To enable processes to dynamically establish and retrieve their priority levels. This improvement makes it possible for the operating system to prioritize operations according to their urgency or significance through a more adaptable and responsive scheduling mechanism. Higher-priority processes can proactively interrupt and take precedence over lower-priority ones thanks to this scheduler, which chooses which process to execute next depending on its priority. This design makes certain that urgent or important activities are completed on time, which results in an efficient use of system resources.

Project ensures effective and flexible use of system resources by improving the current process scheduling mechanism through the introduction of a priority-based approach. An operating system that is more responsive and adaptable is made possible by the expanded system calls, preemptive priority scheduler, and changed PCB structure.

Contents

1	INTRODUCTION	4
1.1	XV6	4
1.2	WSL	5
1.3	Implementing a new scheduling algorithm in xv6	5
1.3.1	Why using QEMU	6
1.3.2	Deliverables	6
1.4	User Applications	6
2	Methodology	6
2.1	Modifying the files in XV6	7
2.2	Add Priority Field:	7
2.3	Implement Priority Scheduler:	8
2.3.1	Preemptive Priority Scheduling:	8
2.3.2	Scheduler Initialization:	9
2.3.3	Context Switching:	9
2.3.4	Implement System Calls:	9
2.3.5	Update Process Execution:	11
3	Results & Screenshots	11
3.1	Testing and Validation	11
3.1.1	Testing Round Robin	15
4	Problems Faced	18
5	Conclusion	18

Problem Statement

Replace the default scheduler of XV6 with a preemptive priority-based scheduler that assigns different priority levels to processes. Processes with higher priority levels should be scheduled more frequently than those with lower priority levels.

OBJECTIVES

- Improve the overall responsiveness of the operating system by introducing a priority-based scheduling mechanism.
- Adjust priorities dynamically to improve process responsiveness to changing workloads and system circumstances.
- Ensure a seamless integration of priority-related enhancements into the existing operating system architecture.
- Implement a preemptive priority scheduler to optimize system resource utilization and response to critical tasks.

Software Used :

- Linux
- WSL
- Qemu

1 INTRODUCTION

1.1 XV6

The xv6 operating system is a lightweight and educational OS developed at MIT. It aims to provide students with hands-on experience in understanding the fundamental concepts that form the basis of modern operating systems. Named after its predecessor, the Sixth Edition Unix operating system, xv6 is a reimplementation that carefully preserves key design principles and features of Unix. Through its simple and organized codebase, xv6 offers an approachable way for students and enthusiasts to explore essential aspects of operating systems, including process management, memory allocation, file systems, and system calls. By studying xv6, individuals can effectively bridge the between theoretical knowledge and practical implementation, making it a tool for comprehending the intricate internals of operating systems. The latest xv6 source code is available at

`git://github.com/mit-pdos/xv6-public.git`

1.2 WSL

Windows Subsystem for Linux (WSL) is a feature of Windows that allows developers to run a Linux environment without the need for a separate virtual machine or dual booting.

There are two versions of WSL: WSL 1 and WSL 2. We used WSL 2 to complete this Project.

WSL 2 includes a full Linux kernel, which runs alongside the Windows kernel. This allows for better compatibility and performance compared to the earlier version of WSL. WSL 2 supports a wide range of Linux distributions available from the Microsoft Store. Users can choose and install their preferred Linux distribution to use alongside their Windows environment. WSL 2 allows us to run Linux commands from Windows, making it easier to work with both Windows and Linux tools in a unified environment.

Following commands Involved in compiling code, managing dependencies and running simulations or tests;

- Removes compiled files and artifacts:
make clean
- Builds the project:
make
- Runs the operating system in a simulated environment:
make qemu or make qemu-nox

1.3 Implementing a new scheduling algorithm in xv6

The current scheduler in xv6 is based on a round-robin scheduling algorithm. We will modify the scheduler to take into account user-defined process priorities by following these steps,

- Adding a new system call to xv6 to set process priorities. When a process calls priority, the priority of the process should be set to the specified value. The priority can be any integer value, with higher values denoting more priority.
- Modify the xv6 scheduler to use this priority in picking the next process to schedule. The only requirement is that a higher numerical priority should translate into more CPU time for the process.

Here is a simple test case to give you an example of what is expected. You can create two identical

1.3.1 Why using QEMU

Testing xv6 directly within WSL without QEMU might be challenging as xv6 is designed to run on a simplified hardware simulator. The absence of a suitable simulator or hardware emulation environment within WSL could make it difficult to execute xv6 directly. However, we can still compile and modify the xv6 source code in WSL within the xv6 code base.

1.3.2 Deliverables

Following are the functionalities provided by project

- Modified Process Control Block (PCB) Structure
- Priority Scheduler Implementation
- Test Case
- Extended System Calls

1.4 User Applications

- Prototyping and Development
- Teaching System Call Interface
- Research and Experimentation
- Educational Purpose

2 Methodology

Procedure

Following steps were taken complete the task;

- First we Locate the definition of struct proc in the proc.h file. This structure holds information about each process in xv6.
- Modify the allocproc function in proc.c to initialize the new fields in the struct proc when a new process is created.
- If necessary, update the fork and exec functions in proc.c to properly initialize the new fields for child processes.

- Added fields related to process priority, for example, modify the process scheduler (scheduler function in `proc.c`) to take those priorities into account when selecting the next process to run.
- Add system calls via `syscall.c` and `syscall.h` files for this.
- Lastly, We created a user-program to verify the working of the scheduler.

2.1 Modifying the files in XV6

In xv6, the scheduler is primarily defined in a few key files that handle process scheduling. To replace the default scheduler, you'll need to make changes primarily in the following files:

1. **proc.h and proc.c** Implements process-related functionalities including process creation, scheduling, and management.
2. Implement your custom scheduling algorithm in functions like `scheduler()` or any other related scheduling functions within `proc.c`.
3. Update function prototypes and definitions related to the scheduler to accommodate changes made in `proc.h` and `proc.c`
4. `syscall.c`, `syscall.h` These both files are edited for the system calls addition.

2.2 Add Priority Field:

- Introduce a priority field in the PCB structure which is defined in `proc.h` to store the priority level of each process.

2.3 Implement Priority Scheduler:

```
void
scheduler(void)
{
    struct proc *p, *p1;
    struct cpu *c = mycpu();
    c->proc = 0;
    for(;;){
        sti();
        struct proc *highP;
        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->state != RUNNABLE)
                continue;
            highP = p;
            for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
                if(p1->state != RUNNABLE)
                    continue;
                if(highP->priority > p1->priority)
                    highP = p1;
            }
            p = highP;
            c->proc = p;
            switchvm(p);
            p->state = RUNNING;
            swtch(&(c->scheduler), p->context);
            switchkvm();
            c->proc = 0;
        }
        release(&ptable.lock);
    }
}
```

Figure 1: Preemptive Priority Scheduler Code

2.3.1 Preemptive Priority Scheduling:

- Developing a priority scheduler algorithm that selects the next process to run based on its priority.
- Preemptive approach is adopted, allowing higher-priority processes to interrupt lower-priority ones.

2.3.2 Scheduler Initialization:

- Initialize the scheduler with an appropriate data structure for managing the ready queue based on priority.

2.3.3 Context Switching:

- Implement context-switching routines to save and restore the state of processes during preemption.

2.3.4 Implement System Calls:

- Introduce new system calls to allow processes to set and retrieve their priority levels dynamically.

syscall.c code:

```

Select root@DESKTOP-D5AU9PO: /home/xv6-public
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_cps(void);
extern int sys_chpr(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_cps]     sys_cps,
[SYS_chpr]    sys_chpr,
};

void
syscall(void)
{
    int num;
    struct proc *curproc = myproc();

    num = curproc->tf->eax;
    if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
        curproc->tf->eax = syscalls[num]();
    } else {
        cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
        curproc->tf->eax = -1;
    }
}

```

Figure 2: syscall.h

syscall.h code:

```

#define SYS_cps      22
#define SYS_chpr     23

```

Figure 3: Caption

We have added these two system calls in the preemptive priority Scheduler.

2.3.5 Update Process Execution:

- Modify the system calls handling routine to incorporate the new priority-related system calls
- Ensure that the scheduler is appropriately informed of priority changes.

3 Results & Screenshots

3.1 Testing and Validation

Test Outputs:

```
root@DESKTOP-DSAU9PO: /home/xv6-public
objdump -t wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > wc.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o zombie.o zombie.c
ld -m elf_i386 -N -e main -ltext 0 -o _zombie zombie.o ulib.o usys.o printf.o umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o ps.o ps.c
ld -m elf_i386 -N -e main -ltext 0 -o _ps ps.o ulib.o usys.o printf.o umalloc.o
objdump -S _ps > ps.asm
objdump -t _ps | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ps.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o dpro.o dpro.c
ld -m elf_i386 -N -e main -ltext 0 -o _dpro dpro.o ulib.o usys.o printf.o umalloc.o
objdump -S _dpro > dpro.asm
objdump -t _dpro | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > dpro.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o nice.o nice.c
ld -m elf_i386 -N -e main -ltext 0 -o _nice nice.o ulib.o usys.o printf.o umalloc.o
objdump -S _nice > nice.asm
objdump -t _nice | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > nice.sym
./mkfs fs.img README cat echo forktest grep init kill ln ls mkdir rm sh stressfs usertests wc _zombie _ps _dpro _nice
mmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
ballocc: first 734 blocks have been allocated
ballocc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.          1 1 512
..         1 1 512
README    2 2 2286
cat        2 3 15500
echo       2 4 14376
forktest   2 5 8820
grep       2 6 18344
init       2 7 15000
kill       2 8 14464
ln         2 9 14360
ls         2 10 16932
mkdir      2 11 14484
rm         2 12 14464
sh         2 13 28524
stressfs   2 14 15396
usertests  2 15 62900
wc         2 16 15920
```

Figure 4: Make QEMU Running

```
exec: fail
exec fork failed
$ echo
$ fork
exec: fail
exec fork failed
$ usertests
usertests starting
arg test passed
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
fourfiles ok
sharedfd test
sharedfd ok
bigarg test
bigarg test ok
bigwrite test
bigwrite ok
bigarg test
bigarg test ok
bss test
bss test ok
sbrk test
pid 101 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80000000--kill proc
pid 102 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8000c350--kill proc
pid 103 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800186a0--kill proc
pid 104 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800249f0--kill proc
pid 105 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80030d40--kill proc
pid 106 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8003d090--kill proc
pid 107 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800493e0--kill proc
pid 108 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80055730--kill proc
pid 109 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80061a80--kill proc
pid 110 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8006ddd0--kill proc
pid 111 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8007a120--kill proc
pid 112 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80086470--kill proc
pid 113 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800927c0--kill proc
pid 114 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8009eb10--kill proc
pid 115 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800aae60--kill proc
pid 116 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800b71b0--kill proc
pid 117 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800c3500--kill proc
pid 118 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800cf850--kill proc
pid 119 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800dbba0--kill proc
pid 120 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800e7ef0--kill proc
```

Figure 5: Test cases running

root@DESKTOP-D5AU9PO: /home/xv6-public

```
pid 112 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80086470--kill proc
pid 113 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800927c0--kill proc
pid 114 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8009eb10--kill proc
pid 115 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800aae60--kill proc
pid 116 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800b71b0--kill proc
pid 117 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800c3500--kill proc
pid 118 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800cf850--kill proc
pid 119 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800dbba0--kill proc
pid 120 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800e7ef0--kill proc
pid 121 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x800f4240--kill proc
pid 122 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80100590--kill proc
pid 123 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8010c8e0--kill proc
pid 124 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80118c30--kill proc
pid 125 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80124f80--kill proc
pid 126 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801312d0--kill proc
pid 127 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8013d620--kill proc
pid 128 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80149970--kill proc
pid 129 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80155cc0--kill proc
pid 130 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80162010--kill proc
pid 131 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8016e360--kill proc
pid 132 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8017a6b0--kill proc
pid 133 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80186a00--kill proc
pid 134 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x80192d50--kill proc
pid 135 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x8019f0a0--kill proc
pid 136 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801ab3f0--kill proc
pid 137 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801b7740--kill proc
pid 138 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801c3a90--kill proc
pid 139 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801cfd00--kill proc
pid 140 usertests: trap 14 err 5 on cpu 0 eip 0x3069 addr 0x801dc130--kill proc
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
allocuvm out of memory
sbrk test OK
validate test
validate ok
open test
open test ok
small file test
creat small succeeded; ok
writes ok
open small succeeded ok
read succeeded ok
small file test ok
big files test
big files ok
```

Figure 6: Validating Test cases running

```
root@DESKTOP-D5AU9PO: /home/xv6-public
allocvm out of memory
sbrk test OK
validate test
validate ok
open test
open test ok
small file test
creat small succeeded; ok
writes ok
open small succeeded ok
read succeeded ok
small file test ok
big files test
big files ok
many creates, followed by unlink test
many creates, followed by unlink; ok
openiput test
openiput test ok
exitiput test
exitiput test ok
iput test
iput test ok
mem test
allocvm out of memory
mem ok
pipe1 ok
preempt: kill... wait... preempt ok
exitwait ok
rmdot test
rmdot ok
fourteen test
fourteen ok
bigfile test
bigfile test ok
subdir test
subdir ok
linktest
linktest ok
unlinkread test
unlinkread ok
dir vs file
dir vs file OK
empty file name
empty file name OK
fork test
fork test OK
bigdir test
bigdir ok
uio test
pid 595 usertests: trap 13 err 0 on cpu 0 eip 0x35c3 addr 0x801dc130--kill proc
```

Figure 7: Last test cases are running

```
uio test
pid 595 usertests: trap 13 err 0 on cpu 0 eip 0x35c3 addr 0x801dc130--kill proc
uio test done
exec test
ALL TESTS PASSED
$
```

Figure 8: All tests are tested

3.1.1 Testing Round Robin

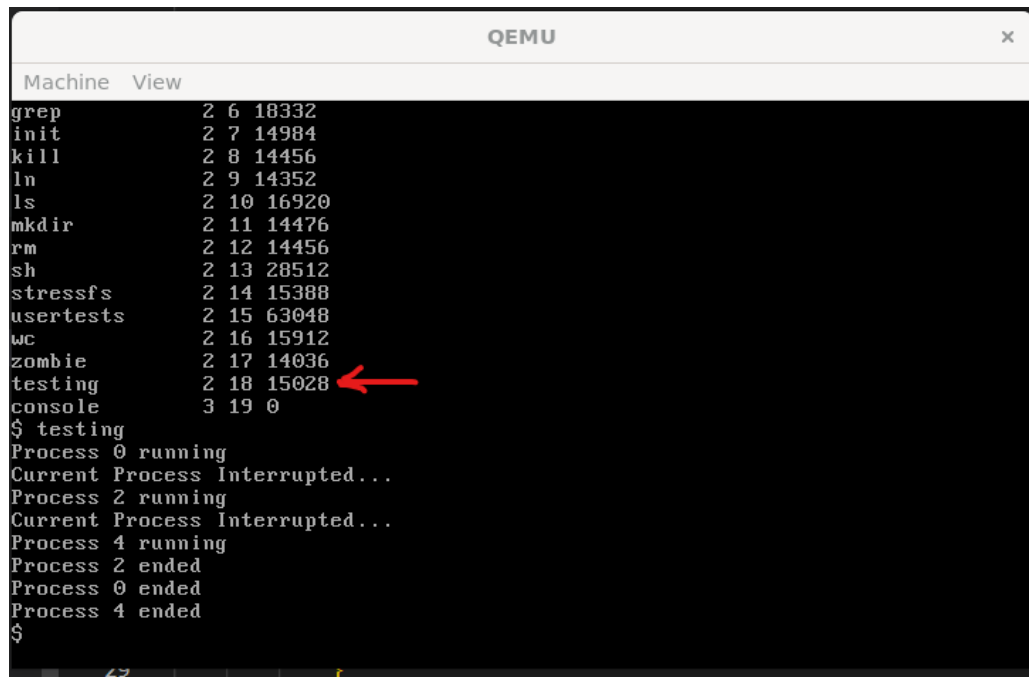
Here's a general guide on how you might approach testing the round-robin scheduler: **Understanding the Scheduler:**

we located the scheduling code within the xv6 source code, named something like scheduler() or sched(). Identify the scheduling algorithm currently implemented in xv6. The round-robin scheduler might be implemented as a part of the scheduling logic. **Testing the Round-Robin Scheduler:**

Once you've implemented or modified the scheduler, you can test its behavior. Create multiple processes or threads that perform some tasks or computations (e.g., simple loops) and observe their execution order and time slices. Check if the scheduler is indeed providing fair time slices to each process in a round-robin fashion. **Running the Test:**

Run xv6 using QEMU as you did before: make qemu Once inside the xv6 environment, execute the modified or implemented scheduler and the test processes you've created. Monitor the execution of processes to see if they're being scheduled in a round-robin manner. **Verify & Observing Results:**

Ensure that each process gets a fair share of CPU time. If multiple processes are running simultaneously, they should switch after the specified time quantum (as defined in the scheduler logic). Check the output or logging mechanisms within xv6 to observe the scheduling behavior. You might need to add print statements or debugging tools within the scheduler code to track process scheduling and execution.



```
QEMU x
Machine View
grep      2 6 18332
init      2 7 14984
kill      2 8 14456
ln        2 9 14352
ls        2 10 16920
mkdir     2 11 14476
rm        2 12 14456
sh        2 13 28512
stressfs  2 14 15388
usertests 2 15 63048
wc        2 16 15912
zombie    2 17 14036
testing   2 18 15028 ←
console   3 19 0
$ testing
Process 0 running
Current Process Interrupted...
Process 2 running
Current Process Interrupted...
Process 4 running
Process 2 ended
Process 0 ended
Process 4 ended
$
```

Analyzing and Refining:

Analyze the results to determine if the round-robin scheduling is working as expected. Refine the scheduler code if necessary, making adjustments to achieve the desired scheduling behavior.

Below figure represents Priority Scheduling:

```

init: starting sh
$ ps
name      pid      state  priority
init      1        SLEEPING    2
sh        2        SLEEPING    2
ps        3        RUNNING    2
$ dpro &; dpro &;
$ Parent 6 creating child 9
hiParent 8 creating child 10
Child 10 crldeated
 9 created
ps
name      pid      state  priority
init      1        SLEEPING    2
sh        2        SLEEPING    2
dpro      9        RUNNABLE   10
dpro     10        RUNNING    10
dpro      6        SLEEPING    2
dpro      8        SLEEPING    2
ps        11       RUNNING    2
$ nice 9 1
$ ps
name      pid      state  priority
init      1        SLEEPING    2
sh        2        SLEEPING    2
dpro      9        RUNNING    1
dpro     10        RUNNABLE   10
dpro      6        SLEEPING    2
dpro      8        SLEEPING    2
ps        13       RUNNING    2
$

```

Figure 9: dpro file running having all processes

4 Problems Faced

1. Synchronization: Managing access to shared data structures, such as process queues and priority levels, is a critical challenge to prevent race conditions.
2. Codebase Understanding: Implementing a priority scheduler requires a deep understanding of the existing xv6 codebase and the interactions between different components.
3. Compatibility: Modifications must be made while ensuring compatibility with other features of xv6 to maintain a stable and functional operating system.
4. Trade-off Consideration: Balancing responsiveness and fairness in process execution is challenging, especially considering the limited resources and simplicity of xv6.
5. Integration: Efficiently integrating the priority-based scheduling algorithm within the design principles and constraints of xv6 poses a non-trivial engineering challenge.

5 Conclusion

To sum up, this project has effectively tackled its goal of putting in place a priority scheduler for the xv6 operating system, exposing students to fundamental ideas in operating system scheduling. The original xv6 scheduler was replaced by a more advanced priority-based scheduler, which treats all processes equally and uses a round-robin method. This new scheduler facilitates a scheduling technique where higher-priority processes are given precedence over lower-priority ones by assigning varying priority levels to processes.

The project was organized around a number of major tasks, such as adding a priority field to the process control block (PCB) structure, extending system calls to manage process priorities dynamically, implementing a preemptive priority scheduler, and creating thorough test cases to assess the effectiveness and correctness of the new scheduler. To evaluate the efficacy of the priority-based strategy, a thorough comparison with the initial round-robin scheduler was conducted throughout the testing phase.

Throughout the project, students engaged in hands-on experience, gaining practical insights into the complexities of operating system scheduling. By delving into the modification of the xv6 operating system and enhancing its scheduler, participants developed a deeper understanding of the fundamentals underlying real-world operating systems.